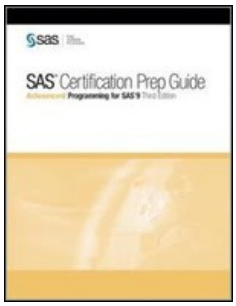


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

Chapter 9: Introducing Macro Variables

Overview

Introduction

SAS macro variables enable you to substitute text in your SAS programs. Macro variables can supply a variety of information, including

- operating system information
- SAS session information
- text strings.

When you reference a macro variable in a SAS program, SAS replaces the reference with the text value that has been assigned to that macro variable. By substituting text into programs, SAS macro variables make your programs more reusable and dynamic.

The following sample code shows how a macro variable might be used to substitute a year value throughout a program, enabling you to quickly and easily change the value of **year** throughout the program:

```
%let year=2002;
proc print data=sasuser.schedule;
   where year(begin_date) = &year;
   title "Scheduled Classes for &year";
run;
proc means data=sasuser.all sum;
   where year(begin_date) = &year;
   class location;
   var fee;
   title1 "Total Fees for &year Classes";
   title2 "by Training Center";
run;
```

Objectives

In this chapter, you learn to

- recognize some benefits of using macro variables
- substitute the value of a macro variable anywhere in a program
- identify and display automatic macro variables
- create and display user-defined macro variables
- recognize how macro variables are processed
- display macro variable values and other text in the SAS log
- use macro character functions
- combine macro references with adjacent text or with other macro variable references
- use macro quoting functions.

Basic Concepts

Overview

In the SAS programs that you write, you might find that you need to reference the same variable, data set, or text string multiple times.

```
title "Total Sales for  2002";
```

```
data perm.sales 2002;
   set perm.sales;
   if year(enddate) =  2002;
run;
proc print data=perm.sales 2002;
run;
```

Then, you might need to change the references in your program in order to reference a different variable, data set, or text string. Especially If your programs are lengthy, scanning for specific references and updating them manually can take a lot of time, and it is easy to overlook a reference that needs to be updated.

```
title "Total Sales for 2001";
data perm.sales2001;
   set perm.sales;
   if year(enddate) = 2002;
run;
proc print data=perm.sales 2001;
run;
```

If you use a *macro variable* in your program, these updates are quick and easy because you need to make the change in only one place.

```
%let year= 2002;
title "Total Sales for &year";
data perm.sales&year;
   set perm.sales;
   if year(enddate)=&year;
run;
proc print data=perm.sale£&year;
run;
```

The value of the macro variable is inserted into your program, so you can make one change and have the change appear throughout the program.

Macro Variables

Macro variables are part of the *SAS macro facility*, which is a tool for extending and customizing SAS and for reducing the amount of program code you must enter in order to perform common tasks. The macro facility has its own language, which enables you to package small or large amounts of text into units that have names. From then on, you can work with the names rather than with the text itself.

There are two types of macro variables:

- *automatic macro variables*, which are provided by SAS
- *user-defined macro variables*, whose values you create and define.

Whether automatic or user-defined, a macro variable is independent of a SAS data set and contains one text string value that remains constant until you change it. The value of a macro variable is substituted into your program wherever the macro variable is referenced.

The value of a macro variable is stored in a *symbol table*. The values of automatic macro variables are always stored in the *global symbol table*, meaning that these values are always available in your SAS session. The values of user-defined macro variables are often stored in the global symbol table as well.

```
%let city=Dallas;
%let date=05JAN2000;
%let amount=975;
```

Global symbol table		
SYSTIME	*09:31*	*←automatic variable*
SYSVER	*9.2*	
CITY	*Dallas*	*←user-defined variable*
DATE	*05JAN2000*	

AMOUNT	*975*	

Macro variables can be defined and referenced anywhere in a SAS program except within the data lines of a DATALINES statement. You will learn more about how to define and reference macro variables throughout this chapter.

Referencing Macro Variables

In order to substitute the value of a macro variable in your program, you must *reference* the macro variable. A macro variable reference is created by preceding the macro variable name with an *ampersand (&)*. The reference causes the *macro processor* to search for the named variable in the symbol table and to return the value of the variable if the variable exists. If you need to reference a macro variable within quotation marks, such as in a title, you must use double quotation marks. The macro processor will not resolve macro variable references that appear within single quotation marks.

Note You will learn more about the macro processor later in this chapter.

Global Symbol Table	
CITY	*Dallas*
DATE	*05 JAN2000*
AMOUNT	*975*

Example: Referencing a Macro Variable

To reference the macro variable **amount** from the global symbol table that is represented above, you place **&amount** in your program, as follows:

```
data new;
   set perm.mast;
   where fee; &amount;
run;
proc print;
run;
```

Code After Substitution

```
data new;
   set perm.mast;
   where fee;975;
run;
proc print;
run;
```

Note You will see representations of code after substitution throughout this chapter. In a SAS session, you will not see this code. These representations are meant to show you what happens to your code behind the scenes, after macro processing.

Example: Referencing a Macro Variable in a Title

To reference the macro variable **city** in a title, you must use double quotation marks to enclose the title text in the TITLE statement, as follows:

```
title "Students from &city";
```

When the macro processor cannot resolve a macro variable reference, a message is printed in the *SAS log*. For example, referencing a nonexistent macro variable results in a warning message. Referencing an invalid macro variable name results in an error message.

Table 9.1: SAS Log

```
34 title "Students from &cityst";
WARNING: Apparent symbolic reference CITYST not resolved.
35
36 title "Students from "the_city_in_which_the_student_is_located";
ERROR: Symbolic variable name THE_CITY_IN_WHICH_THE_STUDENT_I must
       be 32 or fewer characters long.
```

Using Automatic Macro Variables

Overview

SAS creates and defines several *automatic macro variables* for you. Automatic macro variables contain information about your computing environment, such as the date and time of the session, and the version of SAS you are running. These automatic macro variables

- are created when SAS is invoked
- are global (always available)
- are usually assigned values by SAS
- can be assigned values by the user in some cases.

Some automatic macro variables have fixed values that are set when SAS is invoked.

Name	Value
SYSDATE	the date of the SAS invocation (DATE7.)
SYSDATE9	the date of the SAS invocation (DATE9.)
SYSDAY	the day of the week of the SAS invocation
SYSTIME	the time of the SAS invocation
SYSENV	FORE (interactive execution) or BACK (non interactive or batch execution)
SYSSCP	an abbreviation for the operating system that is being used, such as WIN or LINUX
SYSVER	the release of SAS that is being used
SYSJOBID	an identifier for the current SAS session or for the current batch job (the user ID or job name for mainframe systems, the process ID (PID) for other systems)

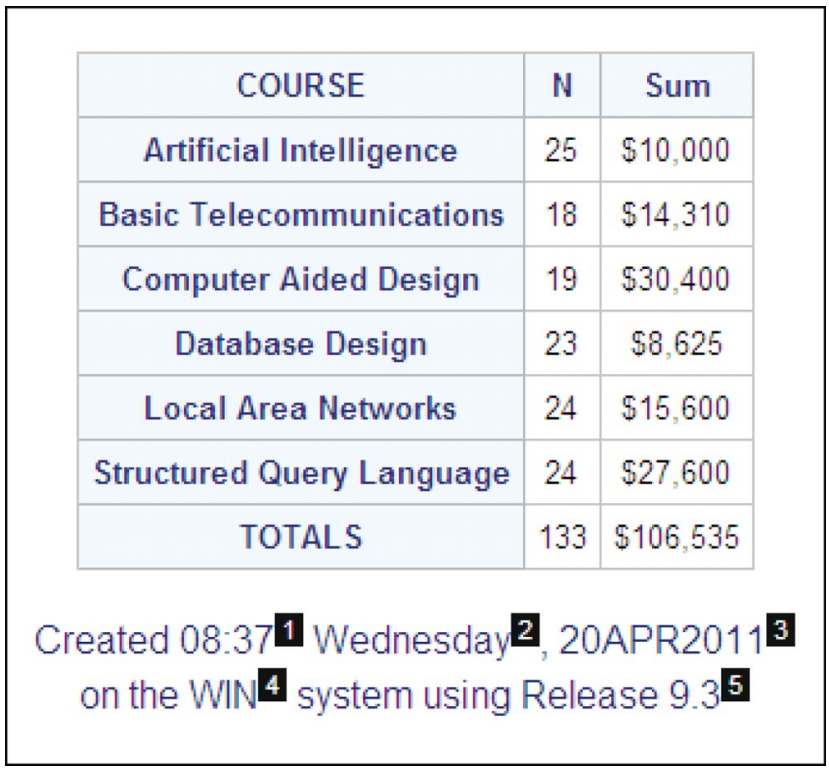
Some automatic macro variables have values that automatically change based on submitted SAS statements.

Name	Value
SYSLAST	the name of the most recently created SAS data set, in the form **LIBREF.NAME.** This value is always stored in all capital letters. If no data set has been created, the value is_NULL_
SYSPARM	text that is specified when SAS is invoked
SYSERR	contains a return code status that is set by the DATA step and some SAS procedures to indicate whether the step or procedure executed successfully

Example

You can substitute system information such as the time, day, and date your SAS session was invoked and the version of SAS you are running in footnotes for a report.

```
footnote1 "Created &systime &sysday, &sysdate9";
footnote2 "on the &sysscp system using Release &sysver";
title "REVENUES FOR DALLAS TRAINING CENTER";
proc tabulate data=sasuser.all(keep=location course_title fee);
   where upcase(location)="DALLAS";
         class course_title;
         var fee;
         table course_title=" " all="TOTALS",
               fee=" "*(n*f=3. sum*f=dollar10.)
               / rts=30 box="COURSE";
run;
```



COURSE	N	Sum
Artificial Intelligence	25	$10,000
Basic Telecommunications	18	$14,310
Computer Aided Design	19	$30,400
Database Design	23	$8,625
Local Area Networks	24	$15,600
Structured Query Language	24	$27,600
TOTALS	133	$106,535

Created 08:37[1] Wednesday[2], 20APR2011[3]
on the WIN[4] system using Release 9.3[5]

1. time of day **(SYSTIME)**
2. day of the week **(SYSDAY)**
3. date (day, month, and year) **(SYSDATE9)**
4. operating environment **(SYSSCP)**
5. release of SAS **(SYSVER)**

Using User-Defined Macro Variables

The %LET Statement

You have seen that SAS provides a variety of automatic macro variables for you. You can also create your own macro variables.

The simplest way to define your own macro variables is to use a *%LET statement*. The %LET statement enables you to define a macro variable and to assign a value to it.

General form, %LET statement:

%LET *variable=value;*

where

variable
is any name that follows the SAS naming convention.

value
can be any string from 0 to 65,534 characters.

variable or *value*
if either contains a reference to another macro variable (such as **&macvar)**, the reference is evaluated before the

assignment is made.

Note *If variable* already exists, *value* replaces the current value.

Example

To create a macro variable named **time** and assign a value of *afternoon* to it, you would submit the following %LET statement:

```
%let time=afternoon;
```

When you use the %LET statement to define macro variables, you should keep in mind the following rules:

- All values are stored as character strings.
- Mathematical expressions are *not* evaluated.
- The case of the value is preserved.
- Quotation marks that enclose literals are stored as part of the value.
- Leading and trailing blanks are removed from the value before the assignment is made.

%LET Statement Examples

When you define a macro variable, remember that its value is always a character string. This table provides examples of macro variable assignment statements to illustrate the rules that are listed in the previous section.

%LET Statement	Variable Name	Variable Value	Length
`%let name= Ed Norton;`	`name`	*Ed Norton*	9
`%let name2=' Ed Norton ';`	`name2`	*'EdNorton '*	13
`%let title="Joan's Report";`	`title`	*"Joan's Report"*	15
`%let start=;`	`start`		0
`%let total=0;`	`total`	*0*	1
`%let sum=4+3;`	`sum`	*4+3*	3
`%let total=&total+&sum`	`total`	*0+4+3*	5
`%let x=varlist;`	`x`	*varlist*	7
`%let &x=name age height;`	`varlist`	*name age height*	15

In the following example, the value *DALLAS* is assigned to the macro variable **site.** The macro variable **site** is then used to control program output.

```
%let site=DALLAS;
title "REVENUES FOR &site TRAINING CENTER";
proc tabulate data=sasuser.all(keep=location
                          course_title fee);
   where upcase(location)='&site";
   class course_title;
   var fee;
   table course_title=' ' all='TOTALS',
         fee=' '*(n*f=3. sum*f=dollar10.)
         / rts=3 0 box='COURSE';
run;
```

REVENUES FOR DALLAS TRAINING CENTER		
COURSE	N	Sum
Artificial Intelligence	25	$10.000
Basic Telecommunications	18	$14.310

Computer Aided Design	19	$30.400
Database Design	23	$8,625
Local Area Networks	24	$15.600
Structured Query Language	24	$27.600
TOTALS	133	$106.535

Processing Macro Variables

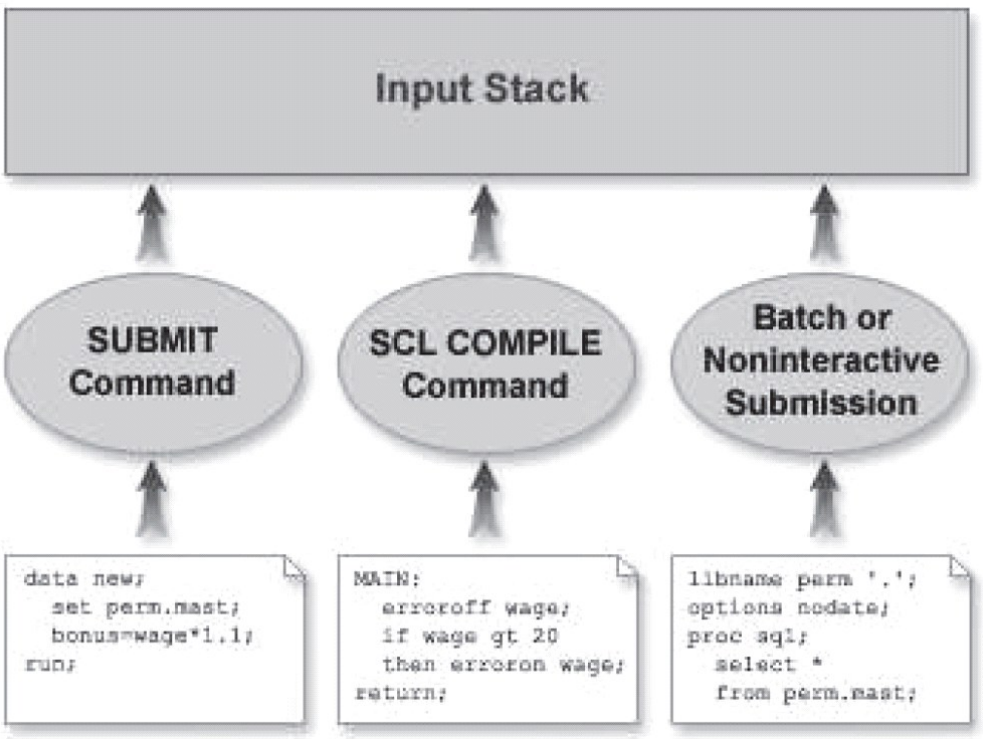
SAS Processing

You have seen how to create and reference macro variables. In order to work with macro variables in the programs that you write, you need to understand how macro variables are processed and stored. First, it is important that you understand how SAS processing works.

A SAS program can be any combination of

- DATA steps and PROC steps
- global statements
- SAS Component Language (SCL) code
- Structured Query Language (SQL) code
- SAS macro language code.

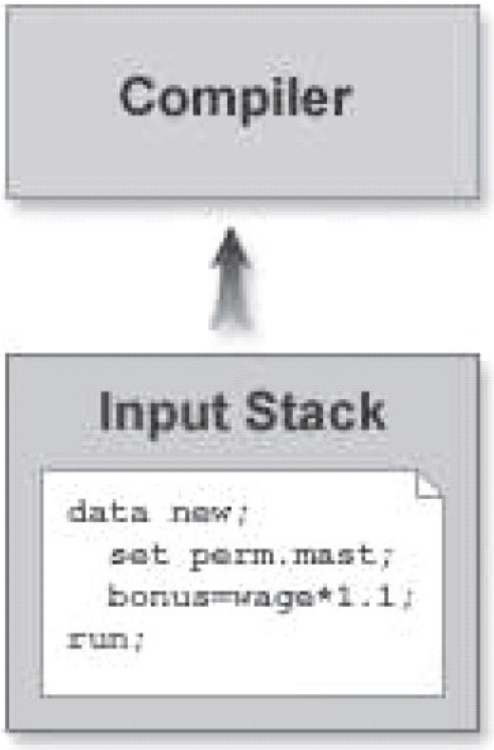
When you submit a program, it goes to an area of memory called the *input stack*. This is true for all code that you submit, such as a DATA step, SCL code, or SQL code.



Once SAS code is in the input stack, SAS

- reads the text in the input stack (left-to-right, top-to-bottom)
- routes text to the appropriate compiler upon demand

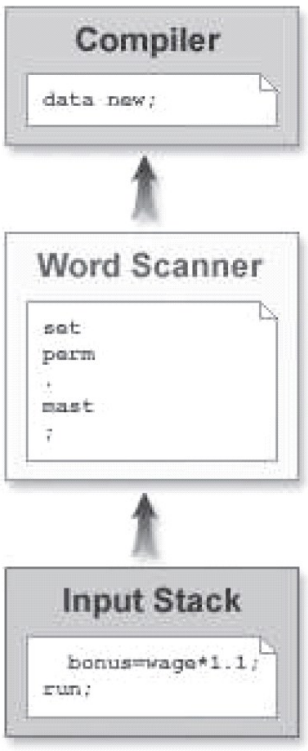
- suspends this activity when a step boundary such as a RUN statement is reached
- executes the compiled code if there are no compilation errors
- repeats this process for any subsequent steps.



Tokenization

Between the input stack and the compiler, SAS programs are *tokenized* into smaller pieces. A component of SAS known as the *word scanner* divides program text into fundamental units called *tokens*.

- Tokens are passed on demand to the compiler.
- The compiler requests tokens until it receives a semicolon.
- The compiler performs a syntax check on the statement.



SAS stops sending statements to the compiler when it reaches a *step boundary*. Examples of step boundaries include a RUN statement **(run;)** or the beginning of a new DATA or PROC step. Once the entire step has been compiled, it is executed.

The word scanner recognizes four types of tokens:

- A *literal* token is a string of characters that are treated as a unit. The string is enclosed in either single or double quotation marks.

  Examples: **"Any text" 'Any text"**

- A *number* token is a string of numerals that can include a period or E-notation (real numbers). Date constants, time constants, date time constants, and hexadecimal constants are also number tokens.

  Examples: **23 109 '01jan2002'd 5e8 42.7**

- A *name* token is a string of characters that begins with a letter or underscore and that continues with underscores, letters, or digits. A period can sometimes be part of a name.

  Examples: **infile _n_ item3 univariate dollar10.2**

- A *special* token is any character or group of characters that has a reserved meaning to the compiler.

  Examples: *** / + - ** ; $ ( ) . & %**

A token ends when the word scanner detects

- the beginning of another token
- a blank after a token.

The maximum length of any token is 32,767 characters.

Examples

- `var x1-x10 z ;`

This example contains six to kens: **`var x1 - x10 z ;`**

- `title 'Report for May';`

  This example contains three to kens: **`title 'Report for May" ;`**

Macro Triggers

Macro variable references and %LET statements are part of the *macro language*. The macro facility includes a macro processor that is responsible for handling all macro language elements. Certain token sequences, known as *macro triggers*, alert the word scanner that the subsequent code should be sent to the macro processor.

The word scanner recognizes the following token sequences as macro triggers:

- % followed immediately by a *name token* (such as *%let)*
- & followed immediately by a *name token* (such as *&amt)*.

When a macro trigger is detected, the word scanner passes it to the macro processor for evaluation. The macro processor

- examines these tokens
- requests additional tokens as necessary
- performs the action indicated.

For macro variables, the processor does one of the following:

- creates a macro variable in the symbol table and assigns a value to the variable
- changes the value of an existing macro variable in the symbol table
- looks up an existing macro variable in the symbol table and returns the variable's value to the input stack in place of the original reference.

The word scanner then resumes processing tokens from the input stack.

Note The word scanner will not recognize macro triggers that are enclosed in single quotation marks. Remember that If you need to reference a macro variable within a literal token, such as the title text in a TITLE statement, you must enclose the text string in *double quotation marks* or the macro variable reference will not be resolved.

Displaying Macro Variable Values in the SAS Log

The SYMBOLGEN Option

When you submit a macro variable reference, the macro processor resolves the reference and passes the value directly back to the input stack. Therefore, you will not see the value that the compiler receives. In order to debug your programs, it might be useful for you to see the value that replaces your macro variable reference. You can use the *SYMBOLGEN* system option to monitor the value that is substituted for a macro variable reference.

General form, OPTIONS statement with SYMBOLGEN option:

OPTIONS NOSYMBOLGEN | SYMBOLGEN;

where

NOSYMBOLGEN
specifies that log messages about macro variable references will not be displayed. This is the default.

SYMBOLGEN
specifies that log messages about macro variable references will be displayed.

This system option displays the results of resolving macro variable references in the SAS log. That is, when the SYMBOLGEN option is turned on, SAS writes a message to the log for each macro variable that is referenced in your program. The message states the macro variable name and the resolved value.

Note Remember that since SYMBOLGEN is a system option, its setting remains in effect until you modify it or until you end your SAS session.

Example

Suppose you have previously assigned values to the macro variables **amount, city**, and **company**, and you submit the following code:

```
data new;
   set sasuser.all;
   where fee>&amount;
   where also city_state contains "&city";
   where also student_company contains '&company';
run;
```

Here is a sample SAS log that shows the messages that are generated by the SYMBOLGEN option for this code. The WHERE ALSO conditions augment the initial WHERE condition using the AND operator. In this example, the where processing will be done for the following condition: (fee>&amount) AND (city state contains "&city") AND (student company contains '&company').

Table 9.2: SAS Log

```
110 where fee>&amount;
SYMBOLGEN: Macro variable AMOUNT resolves to 975
111 where city_state contains "&city";
SYMBOLGEN: Macro variable CITY resolves to Dallas
112 where student_company contains '&company';
```

Notice that no message is displayed for the final macro variable reference **('&company')**. Because this macro variable reference is enclosed in *single quotation marks* rather than in double quotation marks, the word scanner does not call the macro facility to resolve it.

The %PUT Statement

Another way of verifying the values of macro variables is to write your own messages to the SAS log. The %PUT statement writes text to the SAS log.

General form, basic %PUT statement:

%PUT *text;*

where

text
is any text string.

The %PUT statement

- writes only to the SAS log
- always writes to a new log line, starting in column one
- writes a blank line if text is not specified
- does not require quotation marks around text
- resolves macro triggers in text before text is written

- removes leading and trailing blanks from text unless a macro quoting function is used
- wraps lines when the length of text is greater than the current line size setting
- can be used either inside or outside a macro definition.

Example

Suppose you want to verify the value of the macro variable **city.** Since the %PUT statement resolves macro references in text before writing text to the SAS log, you can use it to show the stored value of **city.**

```
%put The value of the macro variable CITY is: &city;
```

Table 9.3: SAS Log

```
120 %put The value of the macro variable CITY is: &city;
The value of the macro variable CITY is: Dallas
```

You can also simply submit the statement **&put &city;** without any additional text. This statement will write the resolved value of the macro variable **city** to the SAS log. However, it will not write any additional text to the log. You might find that it is a good idea to add explanatory text to your %PUT statements in order to maintain clarity in the SAS log. The %PUT statement has several optional arguments that you can add.

Argument	Result in SAS Log
ALL	Lists the values of all macro variables
AUTOMATIC	Lists the values of all automatic macro variables
LOCAL	Lists user-generated local macro variables
USER	Lists the values of all user-defined macro variables

Table 9.4: SAS Log

```
121 %let year=2002;
122 %let city=New York;
123 %let region=South;
124 %put _all_;
GLOBAL YEAR 2002
GLOBAL REGION South
GLOBAL CITY New York
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCC 0
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 29MAY02
```

Notice that when you use optional arguments such as _ALL_, each macro variable name is also written to the SAS log, along with a label of either AUTOMATIC or GLOBAL.

Using Macro Functions to Mask Special Characters

Macro Quoting Functions

The SAS programming language uses matched pairs of either double or single quotation marks to distinguish character constants from names. The quotation marks are not stored as part of the token that they define. For example, in the following program, *var* is stored as a four-byte variable that has the value *text*. If *text* were not enclosed in quotation marks, it would be treated as a variable name. *var2* is stored as a seven-byte variable that has the value *example*.

```
data one;
   var='text';
   text='example';
   var2=text;
run;
```

Similarly, the title text in the following example is **`Joan's Report.`** Although the TITLE statement contains a matched pair of double quotation marks, the title itself does not include these outer quotation marks. However, the outer quotation marks cause the unmatched single quotation mark within the text to be interpreted as an apostrophe that is part of the title text.

```
proc print;
   title "Joan's Report";
run;
```

Example

Earlier you learned that macro variable values are character strings, and you saw examples of macro variables whose values included special characters. Now, suppose you want to store one or more SAS statements in a macro variable. For example, suppose you want to create a macro variable named **`prog`** with *`data new; x=1; run;`* stored as its value.

```
options symbolgen;
%let prog=data new; x=1; run;;
&prog
proc print;
run;
```

Here is part of the SAS log that results from the above program.

Table 9.5: SAS Log

```
25 options symbolgen;
26
27 %let prog=data new;  x=1; run;
27 %let prog=data new;  x=1; run;
                         -
                        180
ERROR 180-322: Statement is not valid or it is used out of proper order.
SYMBOLGEN: Macro variable PROG resolves to data new
28      &prog
29      proc print;
30      run;
NOTE: The data set WORK.NEW has 1 observations and 0 variables.
NOTE: The data set WORK.PROC has 1 observations and 0 variables.
NOTE: The data set WORK.PRINT has 1 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.25 seconds
      cpu time            0.07 seconds
```

Notice that according to the SYMBOLGEN statement in the log, the macro variable **`prog`** has been assigned a value of *data new*. SAS interpreted the first semicolon as the end of the macro assignment statement. In this case, we want the semicolon to be part of the macro variable value, but SAS has no way of knowing that. In this situation, you need to mask text that you want to assign to a macro variable. That is, you need to hide the normal meaning of the semicolon from the macro processor. You can use a *macro quoting function* to do this.

The %STR Function

The %STR function is used to mask (or write quotation marks around) tokens during compilation so that the macro processor does not interpret them as macro-level syntax. That is, the %STR function hides the normal meaning of a semicolon and other special tokens and mnemonic equivalents of comparison or logical operators so that they appear as constant text. Special tokens and mnemonic equivalents include

```
; + - * / , < > = blank A ~ # |
LT EQ GT AND OR NOT LE GE NE IN
```

The %STR function also

- enables macro triggers to work normally
- preserves leading and trailing blanks in its argument.

General form, %STR function:

%STR (*argument)*

where

argument
is any combination of text and macro triggers.

Applying this to our previous example, there are a number of ways that text can be quoted. Remember that we wanted to create a macro variable named **prog** that has *data new; x=l; run;* as its value.

Method One

You could quote all text. **`%let prog=%str(data new; x=1; run;);`**

Method Two

You could quote only the semicolons. **`%let prog=data new%str(;) x=1%str(;)run%str(;);`**

Method Three

You could create an additional macro variable, assign a quoted value to it, and reference it in the assignment statement for the prog macro variable. **`%let s= %str(;); %let prog=data new&s x=1&s run&s;`**

Each of these methods accomplishes the same thing: they all assign the value *data=new; x=l; run;* to the macro variable **prog.**

The %STR function can also be used to quote tokens that typically occur in pairs:

```
' ") (
```

Example

Suppose you want to assign text that contains an apostrophe (') toa macro variable. Without any quoting, this will lead to errors.

```
options symbolgen;
%let text=Joan's Report;
proc print data=sasuser.courses;
   where days > 3;
title "&text";
run;
```

Table 9.6: SAS Log

```
75 %let text=Joan's Report;
                ---------
                32
WARNING 32-169: The quoted string currently being processed has
                become more than 262 characters long. You may
                have unbalanced quotation marks.
```

The word scanner interprets the apostrophe as the beginning of a literal that is defined by a pair of single quotation marks. You can use the %STR function to avoid this error. In the last section you saw several methods of using the %STR function to mask the normal meaning of asemicolon. However, none of the methods shown will correctly mask the apostrophe in our current example.

When you quote tokens that typically appear in pairs, such as quotation marks or parentheses, you must take one

additional step. To perform this quoting, you precede the token that you want to quote with a *percent sign (%)* within the %STR function argument.

```
%let text=%str(Joan%'s Report);
%let text=Joan%str(%')s Report;
```

The value of **text** is *Joan's Report* in both cases.

The %NRSTR Function

Sometimes you might want to hide the normal meaning of an ampersand or a percent sign. The %NRSTR function performs the same quoting function as %STR, except it also masks macro triggers (& and %). The *NR* in the name %NRSTR stands for *No Resolution*. %NRSTR has the same syntax as %STR.

Example

Suppose you want to create a macro variable named *period* and to assign a value of *May&Jun* to it. If you attempt to use the %STR function in the assignment statement, SAS will interpret the ampersand as a macro trigger and generate a warning message. You need to use the %NRSTR function instead.

```
%let Period=%str(May&Jun);
%put Period resolves to: &period;
%let Period=%nrstr(May&Jun);
%put Period resolves to: &period;
```

The following portion of a SAS log shows the results of both the %STR and the %NRSTR functions for this example.

Table 9.7: SAS Log

```
1    %let Period=%str(May&Jun);
WARNING: Apparent symbolic reference JUN not resolved.
2    %put Period resolves to &period:
WARNING: Apparent symbolic reference JUN not resolved.
Period resolves to: May&Jun
3
4    %let Period=%nrstr(May&Jun);
5    %put Period resolves to &period;
Period resolves to: May&Jun
```

The %BQUOTE Function

Like the %STR function, the %BQUOTE function is used to mask (or write quotation marks around) special characters and mnemonic operators. However, while the %STR function performs during compilation, the %BQUOTE function performs during execution. That is, the %BQUOTE function masks a character string or resolved value of a text expression during execution of a macro or macro language statement so that special characters and mnemonic operators are not interpreted as anything other than plain text. Special tokens and mnemonic equivalents include

```
' " ( ) + - * / < > = ¬ A ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

The %BQUOTE function also

- does not require that quotation marks be marked
- enables macro triggers to work normally
- preserves leading and trailing blanks in its argument.

General form, %BQUOTE function:

%BQUOTE (*argument)*

where

argument

is any combination of text and macro triggers.

Example

Remember the example where you want to assign text that contains an apostrophe (')to a macro variable. You used the %STR function to mask the apostrophe.

```
%let text=%str(Joan%'s Report);
%let text=Joan%str(%')s Report;
```

You can accomplish this task using the %BQUOTE function. The %BQUOTE function does not require that unmatched quotation marks be marked, so the title that contains an apostrophe requires no special syntax.

```
%let text=%bquote(Joan's Report);
```

Using Macro Functions to Manipulate Character Strings

Macro Character Functions

Often when working with macro variables, you will need to manipulate character strings. You can do this by using *macro character functions*. With macro character functions, you can

- change lowercase letters to uppercase
- produce a substring of a character string
- extract a word from a character string
- determine the length of a character string, and more.

Macro character functions have the same basic syntax as the corresponding DATA step functions, and they yield similar results. It is important to remember that although they might be similar, macro character functions are distinct from DATA step functions. As part of the macro language, macro functions enable you to communicate with the macro processor in order to manipulate text strings that you insert into your SAS programs. The next few sections explore several macro character functions in greater detail.

The %UPCASE Function

The %UPCASE function enables you to change the value of a macro variable from lowercase to uppercase before substituting that value in a SAS program. Since most comparison operators in the SAS language are case sensitive, it is often necessary to change values to uppercase.

General form, %UPCASE function:

%UPCASE *(argument)*

where

argument

is a character string.

Example

The *Sasuser.All* data set contains student information and registration information for computer training courses. Suppose you want to create a summary of the uncollected course fees:

```
%let paidval=n;
proc means data=sasuser.all sum maxdec=0;
   where paid="&paidval";
   var fee;
```

```
   class course_title;
title "Uncollected Fees for Each Course";
run;
```

Table 9.8: SAS Log

```
163 %let paidval=n;
164 proc means data=sasuser.all sum maxdec=0;
165       where paid="&paidval";
166       var fee;
167       class course_title;
168 title "Uncollected Fees for Each Course";
169 run;

NOTE: No observations were selected from data set SASUSER.ALL.
```

Because the value of the macro variable **paidval** was specified in lowercase, the WHERE expression finds no matching observations. All the values of the data set variable **Paid** are stored in uppercase.

Now we will use the %UPCASE function in the WHERE statement:

```
%let paidval=n;
proc means data=sasuser.all sum maxdec=0;
   where paid="%upcase(&paidval)";
   var fee;
   class course_title;
title "Uncollected Fees for Each Course";
run;
```

You can see that this time the WHERE expression does find matching observations.

Uncollected Fees for Each Course		
The MEANS Procedure		
Analysis Variable : Fee Course Fee		
Description	**N Obs**	**Sum**
Artificial Intelligence	24	9600
Basic Telecommunications	14	11130
Computer Aided Design	13	20800
Database Design	17	6375
Local Area Networks	19	12350
Structured Query Language	20	23000

The %QUPCASE Function

If the argument contains a special character, a mnemonic operator, or a macro trigger, you will need to use the %QUPCASE function. %QUPCASE has the same syntax as the %UPCASE function, and it works the same as %UPCASE except that it also masks mnemonic operators and special characters (including macro triggers) in the function result.

Example

These statements show the results produced by %UPCASE and %QUPCASE:

```
%let a=begin;
%let b=%nrstr(&a);

%put UPCASE produces: %upcase(&b);
%put QUPCASE produces: %qupcase(&b);
```

In the first %PUT statement, the macro reference **&b** resolves to **&a**, which is converted to **&A** because of the %UPCASE function. Since the resolved value contains a macro trigger, it is treated as a macro variable reference and **&A** resolves to the value *begin*. The second %PUT statement uses the %QUPCASE function, which masks the ampersand in the resolved

value of the macro variable **b** so that this value is not treated as another macro variable reference. Executing these statements produces the following messages in the SAS log.

Table 9.9: SAS Log

```
6     %let a=begin;
7     %let b=%nrstr(&a);
8
9     %put UPCASE produces: %upcase(&b);
UPCASE produces: begin
10    %put QUPCASE produces: %qupcase(&b);
QUPCASE produces: &A
```

The %SUBSTR Function

The %SUBSTR function enables you to extract part of a character string from the value of a macro variable.

General form, %SUBSTR function:

%SUBSTR (*argument, position<,n>)*

where

argument

is a character string or a text expression from which a substring will be returned.

position

is an integer or an expression (text, logical, or mathematical) that yields an integer, which specifies the position of the first character in the substring.

n

is an optional integer or an expression (text, logical, or mathematical) that yields an integer that specifies the number of characters in the substring.

Note If the length of *n* is greater than the number of characters following *position* in *argument*, %SUBSTR issues a warning message and returns a substring that contains the characters from *position* to the end of the string. If *n* is not specified, %SUBSTR also returns a substring that contains the characters from *position* to the end of the string.

For example, assume that the macro variable **date** has the value *05JAN2002*.

- Thecode **%substr(&date,3)** will return the value *JAN2002*.
- Thecode **%substr(&date,3,3)** will return the value *JAN*.
- The code **%substr(&date,3,9)** will return the value *JAN2002* and will produce a warning message.

The values of *position* and *n* can also be the result of a mathematical expression that yields an integer. For example, **%substr(&var,%length(&var)-1)** returns the last two characters of the value of the macro variable **var.**

Note The *%LENGTH function* accepts an argument that is either a character string or a text expression. If the argument is a character string, %LENGTH returns the length of the string. If the argument is a text expression, %LENGTH returns the length of the resolved value. If the argument has a null value, %LENGTH returns 0.

Example

Suppose you want to print a report on all courses that have been taught since the start of the current month. You can use the %SUBSTR function and the **SYSDATE9** macro variable to determine the month and year. To start, we need to create an updated class schedule based on the data in **sasuser.schedule**, which is too old for this example:

```
* Update the class schedule based on previous;
data update_schedule;
  set sasuser.schedule;
  begin_date + 3652;
run;
```

Next, we select observations from the updated schedule that are within the current month:

```
* Print a list of courses that started this month;
proc print data=update_schedule;
     where begin_date between
         "01%substr(&sysdate9,3)"d and
         "&sysdate9"d;
   title "All Courses Held So Far This Month";
   title2 "(as of &sysdate9)";
run;
```

All Courses Held So Far This Month
(as of 20APR2011)

Obs	Course_Number	Course_Code	Location	Begin_Date	Teacher
6	6	C006	Boston	02APR2011	Berthan, Ms. Judy

The %QSUBSTR Function

If the argument contains a special character, a mnemonic operator, or a macro trigger, you will need to use the %QSUBSTR function. %QSUBSTR has the same syntax as the %SUBSTR function, and it works the same as %SUBSTR except that it also masks mnemonic operators and special characters (including macro triggers) in the function result.

Example

These statements show the results produced by %SUBSTR and %QSUBSTR:

```
%let a=one;
%let b=two;
%let c=%nrstr(&a &b);

%put C: &c;
%put With SUBSTR: %substr(&c,1,2);
%put With QSUBSTR: %qsubstr(&c,1,2);
```

Executing these statements produces the following messages in the SAS log. As you can see, the first %PUT statement shows that **&c** resolves to the value **&a &b.** In the second %PUT statement, the %SUBSTR function extracts the value **&a** from the resolved value of the macro variable reference **&c**, and resolves **&a** to *one*. The third %PUT statement shows that the %QSUBSTR function prevents the value **&a** from being resolved further.

Table 9.10: SAS Log

```
11   %leta=one;
12   %letb=two;
13   %letc=%nrstr(&a &b);
14
15   %putC: &c;
C: &a &b
16   %putWith SUBSTR: %substr(&c,1,2)
With SUBSTR: one
17   %put With QSUBSTR: %qsubstr(&c,1,2);
With QSUBSTR: &a
```

The %INDEX Function

The %INDEX function enables you to determine the position of the first character of a string within another string.

General form, %INDEX function:

%INDEX *(source, string)*

where

source and *string*

both are character strings or text expressions that can include

- constant text
- macro variable references
- macro functions
- macro calls.

The %INDEX function

- searches *source* for the first occurrence of *string*
- returns a number representing the position in *source* of the first character of *string* when there is an exact pattern match
- returns **0** when there is no pattern match.

Example

The following statements find the first character **v** in a string:

```
%let a=a very long value;
%let b=%index(&a,v);
%put The character v appears at position &b.;
```

Executing these statements writes the following line to the SAS log.

Table 9.11: SAS Log

```
The character v appears at position 3.
```

The %SCAN Function

The %SCAN function enables you to extract words from the value of a macro variable.

General form, %SCAN function:

%SCAN *(argument, n<,delimiters>)*

where

argument
consists of constant text, macro variable references, macro functions, or macro calls.

n
is an integer or a text expression that yields an integer, which specifies the position of the word to return. If *n* is greater than the number of words in *argument*, the functions return a null string.

delimiters
specifies an optional list of one or more characters that separate "words" or text expressions that yield one or more characters.

Caution If *argument* contains a comma, you must enclose *argument* in a quoting function. Similarly, in order to use a single blank or a single comma as the only *delimiter*, you must enclose the character in the %STR function.

The delimiters that %SCAN recognizes vary between ASCII and EBCDIC systems. If you omit delimiters, SAS treats the following characters as *default delimiters'*.

- ASCII systems: **blank . < (+ & ! $ *) ; " - / , % |**
- EBCDIC systems: **blank . < (+ | & ! $ *) ; ¬ - / , % ¦ ¢**

If the delimiter list includes any of the default delimiters for your system, the remaining default delimiters are treated as text.

Example

You can use PROC DATASETS along with the %SCAN function and the **SYSLAST** macro variable to investigate the structure of the most recently created data set:

```
data work.thisyear;
   set sasuser.schedule;
    where year(begin_date) =
          year("&sysdate9"d);
run;

%let libref=%scan(&syslast,1,.);
%let dsname=%scan(&syslast,2,.);
proc datasets lib=&libref nolist;
title "Contents of the Data Set &syslast";
   contents data=&dsname;
run;
quit;
```

Contents of the Data Set WORK,THISYEAR
The DATA SETS Procedure

Data Set Name	WORK.THISYEAR	Observations	0
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Wednesday. April 20. 2011 11:32:11 AM	Observation Length	48
Last Modified	Wednesday. April 20,2011 11:32:11 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin 1 Western (Windows)		

The %QSCAN Function

If the argument contains a special character, a mnemonic operator, or a macro trigger, you will need to use the %QSCAN function. %QSCAN has the same syntax as the %SCAN function, and it works the same as %SCAN except that it also masks mnemonic operators and special characters (including macro triggers) in the function result.

Example

These statements show the results produced by %SCAN and %QSCAN:

```
%let a=one;
%let b=two;
%let c=%nrstr(&a*&b);

%put C: &c;
```

```
%put With SCAN: %scan(&c,1,*);
%put With QSCAN: %qscan(&c,1,*);
```

Executing these statements produces the following messages in the SAS log.

Table 9.12: SAS Log

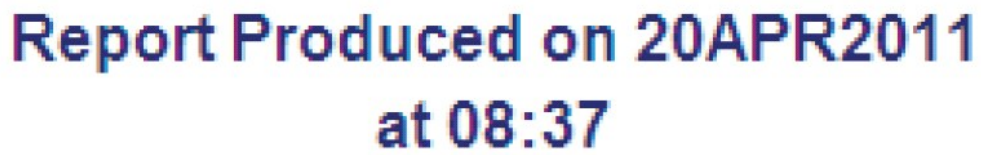
```
47 %leta=one,
48 %letb=two,
49 %letc=%nrstr(&a*&b),
50
51 %putC: &c,
C: &a*&b
52 %putWith SCAN: %scan(&c,1,*);
With SCAN:: one
53 %putWith QSCAN: %qscan(&c,1,*);
With QSCAN: &a
```

Using SAS Functions with Macro Variables

The %SYSFUNC Function

You have learned that by using the automatic macro variables **SYSDATE9** and **SYSTIME** you can include the date and time in a title:

```
titlel "Report Produced on &sysdate9";
title2 "at &systime";
```



SYSDATE9 represents the date on which the SAS session started, and **SYSTIME** represents the time at which the SAS session started. Suppose you would rather see the date in some other format, or suppose you would rather see the current date or time. You can use the %SYSFUNC function to execute other SAS functions as part of the macro facility.

General form, %SYSFUNC function:

%SYSFUNC *(function (argument(s)) <,format>)*

where

function

is the name of the SAS function to execute.

argument(s)

is one or more arguments that are used by *function*. Use commas to separate all arguments. An argument can be a macro variable reference or a text expression that produces arguments for a function.

format

is an optional format to apply to the result of *function*. By default, numeric results are converted to a character string using the BEST12. format, and character results are used as they are, without formatting or translation.

All SAS functions can be used with %SYSFUNC *except*

ALLCOMB	LEXCOMB
ALLPERM	LEXCOMBI
DIF	LEXPERK
DIM	LEXPERM
HBOUND	MISSING
INPUT	PUT
IORCMSG	RESOLVE
LAG	SYMGET
LBOUND	Variable information functions

Note Variable information functions include functions such as VNAME and VLABEL. For a complete list of variable information functions, see "Functions and CALL Routines by Category" in *SAS Functions and CALL Routines: Reference*.

Note You can use the INPUTC or INPUTN function in place of the INPUT function. Similarly, you can use the PUTC or PUTN function in place of the PUT function with %SYSFUNC.

Example

Suppose the following code was submitted on Friday, June 7, 2007:

```
title "%sysfunc(today(),weekdate.) - SALES REPORT";
```

The title on the next report would be **Friday, June 7, 2007 - SALES REPORT**.

Quoting with %QSYSFUNC

As with macro character functions, If the argument for a %SYSFUNC function contains special characters or mnemonic operators, you must use the quoting version of the function. The %QSYSFUNC function has the same syntax as the %SYSFUNC function. %QSYSFUNC works the same as %SYSFUNC except that it also masks mnemonic operators and special characters in the function result.

Example

Suppose you want to create a report title that includes the current date in WORDDATE. format. You could use this statement:

```
title "Report Produced on %sysfunc(today(),worddate.)";
```

However, that would result in the following title:

- **Report Produced on June 7, 2007**

The extra blanks are from the default length of the WORDDATE. format. You need to leftjustify the resulting formatted date. You *cannot* nest functions within %SYSFUNC, but you can use a %SYSFUNC for each function that you need, as shown in this example:

```
title "Report Produced on
             %sysfunc(left(%sysfunc(today(),worddate.)))";
```

However, this statement results in the following error message.

Table 9.13: SAS Log

```
ERROR: The function LEFT referenced by the %SYSFUNC or
       %QSYSFUNC macro function has too many arguments.
```

The LEFT function expects only *one* argument, but you are passing "June 7, 2007" to it. It interprets the comma as the delimiter between *two* arguments.

You can mask the comma by using the %QSYSFUNC function instead, as follows:

```
title "Report Produced on
```

```
%sysfunc(left(%qsysfunc(today(),worddate.)))";
```

The modified statement generates the following title:

- **Report Produced on June 7, 2007**

Combining Macro Variable References with Text

Overview

You can reference macro variables anywhere in your program. Some applications might require placing a macro variable reference adjacent to leading text (text *&variable)* or trailing text *(&variable*text) or referencing adjacent macro variables *(&variable&variable)* in order to build a new token. When you combine macro variable references and text, it is important to keep in mind how SAS interprets tokens.

Remember that a token ends when the word scanner detects either the beginning of a new token or a blank after a token.

You can place text *immediately before* a macro variable reference to build a new token. For example, suppose that data sets are stored in a SAS library, using the naming convention *Yyymon*, where *yy* is a two-digit year such as **02** or **01**, and *mon* is a three-letter month such as **JUN** or **AUG.** Data set names could include examples such as *Y01DEC* and *Y02MAR*. You can write a program that uses a macro variable to build the month portion of the SAS data set name.

```
%let month=jan;
proc chart data=sasuser.y02&month;
   hbar week / sumvar=sale;
run;
proc plot data=sasuser.y02&month;
   plot sale*day;
run;
```

Table 9.14: Code After Substitution

```
proc chart data=sasuser.y02jan;
   hbar week / sumvar=sale;
run;
proc plot data=sasuser.y02jan;
   plot sale*day;
run;
```

Caution If you are using the SAS Learning Edition, you will not be able to submit this code because it uses PROC CHART and the CHART procedure is not included in the software. This example is used in the next several sections and in the chapter summary.

You can reference macro variables that have *no blanks between them* to build new tokens.

For example, you can modify the previous program to enable both the month and the year to be substituted:

```
%let year=02;
%let month=jan;
proc chart data=sasuser.y&year&month;
   hbar week / sumvar=sale;
run;
proc plot data=sasuser.y&year&month;
   plot sale*day;
run;
```

Table 9.15: Code After Substitution

```
proc chart data=sasuser.y02jan;
    hbar week / sumvar=sale;
run;
proc plot data=sasuser.y02jan;
    plot sale*day;
run;
```

The generated program is identical to the program in the previous example. That is, the compiler sees the same code for

both of these examples.

You can place text *immediately after* a macro variable reference as long as the macro variable name can still be tokenized correctly.

For example, you can modify the previous program to substitute the name of an analysis variable:

```
%let year=02;
%let month=jan;
%let var=sale;
proc chart data=sasuser.y&year&month;
   hbar week / sumvar=&var;
run;
proc plot data=sasuser.y&year&month;
   plot &var*day;
run;
```

Table 9.16: Code After Substitution

```
proc chart data=sasuser.y02jan;
   hbar week / sumvar=sale;
run;
proc plot data=sasuser.y02jan;
   plot sale*day
run;
```

The generated program is identical to the program in the previous two examples. That is, although you are changing the code that you submit, you are not changing the code that the compiler sees.

Delimiters in Macro Variable Names

Sometimes you might want to place a macro variable name immediately before text other than a special character. For example, you might want to modify the previous program so that it is easy to switch between using the CHART and PLOT procedures of Base SAS software and the GCHART and GPLOT procedures of SAS/GRAPH software.

```
/* GRAPHICS should be null or G */
%let graphics=g;
%let year=02;
%let month=jan;
%let var=sale;
proc &graphicschart data=sasuser.y&year&month;
   hbar week / sumvar=&var;
run;
proc &graphicsplot data=sasuser.y&year&month;
   plot &var*day;
run;
```

The messages written to the SAS log reveal problems with this program.

Table 9.17: SAS Log

```
13     %let graphics=g;
14     %let year=02;
15     %let month=jan;
16     %let var=sale;
17     proc &graphicschart data=sasuser.y&year&month;
                -
               10
WARNING: Apparent symbolic reference GRAPHICSCHART not resolved.

ERROR 10-205: Expecting the name of the procedure to be executed.
```

SAS interprets the macro variable's name to be **`graphicschart`** instead of **`graphics`** because there is no *delimiter* between the macro variable reference and the trailing text.

The word scanner recognizes the end of a macro variable name when it encounters a special character that cannot be part

of the name token. In other words, the special character acts as a delimiter. For example, a *period (.)* is a special character that is treated as part of the macro variable reference and that does not appear when the macro variable is resolved.

To correct the problem in the previous example, you need to add a period after the reference to the macro variable **graphics.**

```
%let graphics=g;
%let year=02;
%let month=jan;
%let var=sale;
proc &graphics.chart data=sasuser.y&year&month;
    hbar week / sumvar=&var;
run;
proc &graphics.plot data=sasuser.y&year&month;
    plot &var*day;
run;
```

When these SAS statements are executed

- the word scanner treats **&graphics.** as the reference
- the value of the macro variable **graphics** is returned to the input stack
- the word scanner processes **gchart** as one token.

Table 9.18: Code After Substitution

```
proc gchart data=sasuser.y02jan;
   hbar week / sumvar=sale;
run;
proc gplot data=sasuser.y02jan;
   plot sale*day;
run;
```

We can extend this example and further modify the previous program to include a macro variable that is used to define the libref:

```
%let lib=sasuser;
%let graphics=g;
%let year=02;
%let month=jan;
%let var=sale;
libname &lib 'SAS-data-library';
proc &graphics.chart data=&lib.y&year&month;
   hbar week / sumvar=&var;
run;
proc &graphics.plot data=&lib.y&year&month;
   plot &var*day;
run;
```

Notice, however, that this code does not perform the desired substitutions.

Table 9.19: Code After Substitution

```
libname sasuser 'SAS-data-library';
proc gchart data=sasusery02jan;
   hbar week / sumvar=sale;
run;
proc gplot data=sasusery02jan;
   plot sale*day;
run;
```

The period after **&lib** is interpreted as a delimiter. You need to use a second period after the delimiter period to supply the necessary token:

```
%let lib=sasuser;
```

...

```
libname &lib 'SAS-data-library';
proc &graphics.chart data=&lib..y&year&month;
...
proc &graphics.plot data=&lib..y&hear&month;
```

The first period is treated as a delimiter, and the second period is treated as text.

Table 9.20: Code After Substitution

```
proc gchart data=sasuser.y02jan;
…
proc gplot data=sasuser.y02jan;
```

Summary

Contents

This section contains the following topics.

Text Summary

Basic Concepts

Macro variables can supply a variety of information, from operating system information, to SAS session information, to any text string that you define. Updating multiple references to a variable, data set, or text string is a simple process If you use macro variables in your programs. Macro variables are part of the SAS macro facility, which is a tool for extending and customizing SAS and for reducing the amount of text you must enter in order to perform common tasks.

Values of macro variables are stored in symbol tables. Values that are stored in the global symbol table are always available. In order to substitute the value of a macro variable in your program, you must reference that macro variable by preceding the macro variable name with an ampersand. You can reference a macro variable anywhere in a SAS program except within data lines.

Using Automatic Macro Variables

SAS provides automatic macro variables that contain information about your computing environment. Automatic macro variables are created when SAS is invoked. Many of these variables have fixed values that are assigned by SAS and which remain constant for the duration of your SAS session. Others have values that are updated automatically based on submitted SAS statements.

Using User-Defined Macro Variables

You can create and define your own macro variables with the %LET statement. The %LET statement enables you to assign a value for your new macro variable and to store that value in the global symbol table. Macro variable values are character strings; except for leading and trailing blanks, values are stored exactly as they appear in the statement.

Processing Macro Variables

When submitted, a SAS program goes to an area of memory called the input stack. From there, the word scanner divides the program into small chunks called tokens and passes them to the appropriate compiler for eventual execution. Certain token sequences are macro triggers, which are sent to the macro processor for resolution. Once a macro variable has been resolved by the macro processor, the stored value is substituted back into the program in the input stack, and word scanning continues.

Displaying Macro Variable Values in the SAS Log

You can use the SYMBOLGEN system option to monitor the value that is substituted for a macro variable reference. You can also use the %PUT statement to write messages, which can include macro variable values, to the SAS log.

Using Macro Functions to Mask Special Characters

The %STR function enables you to quote tokens during compilation in order to mask them from the macro processor. The %NRSTR function enables you to quote tokens that include macro triggers from the macro processor. The %BQUOTE function enables you to quote a character string or resolved value of a text expression during execution of a macro or macro language statement.

Using Macro Functions to Manipulate Character Strings

You can use macro character functions to apply character string manipulations to the values of macro variables. The %UPCASE function enables you to change values from lowercase to uppercase. The %QUPCASE function works the same as %UPCASE except that it also masks special characters and mnemonic operators in the function result. The %SUBSTR function enables you to extract part of a string from a macro variable value. The %QSUBSTR function works the same as %SUBSTR except that it also masks special characters and mnemonic operators in the function result. The %INDEX function enables you to determine the location of the first character of a character string within a source. Using the %SCAN function, you can extract words from the value of a macro variable. The %QSCAN function works the same as %SCAN except that it also masks special characters and mnemonic operators in the function result.

Using SAS Functions with Macro Variables

You can use the %SYSFUNC function to execute other SAS functions. The %QSYSFUNC function works the same as the %SYSFUNC function except that it also masks special characters and mnemonic operators in the function result.

Combining Macro Variable References with Text

You might sometimes need to combine a macro variable reference with other text. You can place text immediately before or immediately after a macro variable reference. You can also combine two macro variable references in order to create a new token. You might need to use a delimiter when you combine macro variable references with text.

Syntax

```
OPTIONS NOSYMBOLGEN | SYMBOLGEN;
%PUT text;
%LET variable=value;
%STR (argument)
%NRSTR (argument)
%BQUOTE (argument)
%UPCASE (argument)
%QUPCASE (argument)
%SUBSTR (argument, position <,n>)
%QSUBSTR (argument, position <,n>)
%INDEX (source, string)
%SCAN (argument, ? <,delimiters>)
%QSCAN (argument, ? <,delimiters>)
%SYSFUNC (function(argument(s))<,format>)
%QSYSFUNC (function(argument(s))<,format>)
```

Sample Programs

Creating Macro Variables with a %LET Statement

```
options symbolgen;
%let year=2002;
proc print data=sasuser.schedule;
   where year(begin_date)=&year;
   title "Scheduled Classes for &year";
run;
proc means data=sasuser.all sum;
   where year(begin_date)=&year;
   class location;
   var fee;
   title1 "Total Fees for &year Classes";
   title2 "by Training Center";
run;
```

Using Automatic Macro Variables

```
footnote1 "Created &systime &sysday, &sysdate9";
footnote2 "on the &sysscp system using Release &sysver";
title "REVENUES FOR DALLAS TRAINING CENTER";
proc tabulate data=sasuser.all(keep=location course_title fee);
   where upcase(location)="DALLAS";
        class course_title;
        var fee;
        table course_title=" " all="TOTALS",
              fee=" "*(n*f=3. sum*f=dollar10.)
              / rts=30 box="COURSE";
run;
```

Inserting Macro Variables Immediately After Text

```
%let year=02;
%let month=jan;
proc chart data=sasuser.y&year&month;
   hbar week / sumvar=sale;
run;
proc plot data=sasuser.y&year&month;
  plot sale*day;
run;
```

Inserting Macro Variables Immediately Before Text

```
%let graphics=g;
%let year=02;
%let month=jan;
%let var=sale;
proc &graphics.chart data=sasuser.y&year&month;
   hbar week / sumvar=&var;
run;
proc &graphics.plot data=sasuser.y&year&month;
   plot &var*day;
run;
```

Points to Remember

- Macro variables can make your programs more reusable and dynamic.
- When you submit code to SAS, macro variable references are resolved by the macro processor, and their values are substituted into your program.
- You can use the %PUT statement to write any text, including resolved macro variables, to the SAS log.
- If you reference a macro variable within quotation marks, you must use double quotation marks. Macro variable references that are enclosed in single quotation marks will not be resolved.
- Most macro character functions have corresponding functions (such as %QSUBSTR and %QSCAN) that also mask special characters and mnemonic operators in the function result.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which of the following statements is false? ?

a. A macro variable can be defined and referenced anywhere in a SAS program except within data lines.

b. Macro variables are always user-defined, and their values remain constant until they are changed by the user.

c. Macro variables are text strings that are independent of SAS data sets.

d. The values of macro variables can be up to 65,534 characters long.

2. Which of the following TITLE statements correctly references the macro variable **`month?`** ?

a. `title "Total Sales for '&month' ";`

b. `title "Total Sales for 'month'";`

c. `title "Total Sales for &month";`

d. `title Total Sales for "&month";`

3. Which of the following Title statements correctly references the macro variable **`month?`** ?

a. `options &month;`

b. `%PUT &month;`

c. `options symbolgen;`

d. `%PUT the macro variable MONTH has the value &month.;`

4. Which statement will create a macro variable named **`location`** that has the value *storage?* ?

a. `&let location = storage;`

b. `let &location = storage;`

c. `%let location = "storage";`

d. `%let location = storage;`

5. What value will these statements assign to the macro variable **`reptitle:`** ?

```
%let area = "Southeast";
%let reptitle = * Sales Report for &area Area *;
```

a. *Sales Reportfor Southeast Area*

b. *SalesReportfor "Southeast" Area*

c. **SalesReportfor "Southeasf'Area**

d. **SalesReportfor "Southeast" Area **

6. Assuming that you began your SAS session today, which of the following statements correctly sets the macro variable **`currdate`** to today's date: ?

a. `%let currdate = %sysfunc(today(), worddate.);`

b. `%let currdate = &sysdate9;`

c. `%let currdate = %sysfunc(date());`

d. all of the above

7. Macro character functions ?

a. can be used to manipulate character strings in macro variable values.

b. have the same basic syntax as the corresponding DATA step functions and yield similar results.

c. all of the above

d. none of the above

8. The four types of tokens that SAS recognizes are ?

a. expressions, literals, names, and special characters.

b. literals, names, numbers, and special characters.

c. expressions, names, numbers, and special characters.

d. expressions, literals, numbers, and special characters.

9. What are the resulting values for the macro variables that are defined here? ?

```
%let month1 = June;
%let month2 = July;
%let period1 = &month1&month2;
%let period2 = May&month1;
%let period3 = &month2.Aug;
```

a. month1 *junemonth*2 *julyperiod* l *June julyperiod*2 *May June* period3 *July Aug*

b. month1 *junemonth*2 *julyperiod* l *JuneJulyperiod2 MayJune* period3 *July.Aug*

c. month1 *junemonth*2 *julyperiod* l *JuneJulyperiod2 MayJune* period3 *JulyAug*

d. month1 *junemonth*2 *Julyperiod* l *junejulyperiod*2 *Mayjune* period3 *julyaug*

10. Which of the following correctly produces a title in which the current date is left justified in order to remove extra blanks after the word **'for'.?** ?

a. `title "Report for %sysfunc(left(%sysfunc(today(),worddate.)))";`

b. `title "Report for %sysfunc(left(today(), worddate.))";`

c. `title "Report for %sysfunc(left(%qsysfunc(today(), worddate.)))";`

d. `title "Report for %left(today(), worddate.))";`

Answers

1. Correct answer: b

Macro variables are always text strings that are independent of SAS data sets. The value of a macro variable can be up to 65,534 characters long, and the name of a macro variable can be up to 32 characters long. A macro variable can be defined or referenced anywhere in a SAS program except within data lines. There are two types of macro variables: automatic and user-defined.

2. Correct answer: c

To reference a macro variable, you precede the name with an ampersand. You do not need to enclose the macro variable reference in quotation marks.

3. Correct answer: a

There are two ways to display the value of a macro variable in the SAS log: you can tum on the SYMBOLGEN system option to list the values of all macro variables that are used, or you can use the %PUT statement to write specific text, including macro variable values, to the log.

4. Correct answer: d

You use the %LET statement to define a macro variable. You do not need to enclose the value in quotation marks. If you do include quotation marks in the assigned value for a macro variable, the quotation marks will be stored as part of the value.

5. Correct answer: d

Macro variables are stored as character strings. Quotation marks and most special characters are stored exactly as they are assigned, but leading blanks are stripped from assigned values. You can also include references to other macro variables within %LET statements.

6. Correct answer: d

`SYSDATE9` is an automatic macro variable that stores the date that your SAS session began in `ddmmmyyyy` format. You can use the %SYSFUNC function along with any DATA step function, so both the TODAY() function and the DATE() function will result in the current date.

7. Correct answer: c

Macro character functions such as %UPCASE and %SUBSTR enable you to perform character manipulations on your macro variable values.

8. Correct answer: b

The word scanner recognizes four types of tokens. Expressions are not a type of token.

9. Correct answer: c

You can combine macro variable references with text to create new text strings. If you precede a macro variable with text, the ampersand at the beginning of the macro variable name signals the end of the text and the beginning of a macro variable name. If you want text to follow the macro variable value, you must signal the end of the macro variable name with a period.

10. Correct answer: c

You use the %QSYSFUNC function in this case, in order to mask the comma that results from the word date. format. You must mask this comma since the LEFT() function expects only one argument.